EQUATION SOLVING

Cross reference to related applications

This application is a continuation in part of International Application PCT/GB03/001568, filed April 10, 2003, which claims priority to Great Britain Application No. GB 0208329.3, filed April 11, 2002, the contents of each of which are incorporated herein by reference.

Field of Invention

The present invention relates to systems and methods for solving systems of linear equations.

Background of Invention

Systems of linear equations occur frequently in many branches of science and engineering. Effective methods are needed for solving such equations. It is desirable that systems of linear equations are solved as quickly as possible.

A system of linear equations typically comprises N equations in N unknown variables. For example, where N = 3 an example system of equations is set out below:

$$15x + 5y - 2z = 15$$
$$5x + 11y + 4z = 47$$
$$-2x + 4y + 9z = 51$$

In this case, it is necessary to find values of x, y, and z which satisfy all three equations. Many methods exist for finding such values of x, y and z and in this case it can be seen that $x = 1$, $y = 2$ and $z = 5$ is the unique solution to the system of equations.

In general terms, existing methods for solving linear equations can be categorised as either direct methods, or iterative methods. Direct methods attempt to produce an

exact solution by using a finite number of operations. Direct methods however suffer from a problem in that the number of operations required is often large which makes the method slow. Furthermore, some implementations of such methods are sensitive to truncation errors.

Iterative methods solve a system of linear equations by repeated refinements of an initial approximation until a result is obtained which is acceptably close to the accurate solution. Each iteration is based on the result of the previous iteration, and, at least in theory, each iteration should improve the previous approximation. Generally, iterative methods produce an approximate solution of the desired accuracy by yielding a sequence of solutions which converge to the exact solution as the number of iterations tends to infinity.

It will be appreciated, that systems of linear equations are often solved using a computer apparatus. Often, equations will be solved by executing appropriate program code on a microprocessor. In general terms, a microprocessor can represent decimal numbers in fixed point or floating point form.

In fixed point form, a binary number of P bits comprises a first part of length q and a second part of length r. The first part represents the whole number part, and the second part represents the fractional part. In general terms, arithmetic operations can be relatively efficiently implemented for fixed point numbers. However, fixed point numbers suffer from problems of flexibility given that the position of the decimal point is fixed and therefore the range of numbers which can be accurately represented is relatively small given that overflow and round off errors regularly occur.

Floating point numbers again in general terms comprise two parts. A first part, known as the exponent, and a second part known as the mantissa. The mantissa represents the binary number, and the exponent is used to determine the position of the decimal point within that number.

For example considering an eight bit value 00101011 where the first bit represents sign, the subsequent three bits represent the exponent, and the final four bits represent the mantissa, this value is analysed as follows. The first bit represents sign, and is interpreted such that the number is positive if the sign bit is equal to '0' and negative if the sign bit is equal to '1'. In this case, given that the first bit is '0', the number is determined to be positive. The mantissa (1011) is written out and a decimal point is placed to its left side as follows:

.1011

The exponent is then interpreted as an integer, that is 010 is interpreted as the integer value 2 (given that the first bit of the exponent field is a sign bit which determines the direction in which the decimal point should move). In this case, the decimal point is moved to the right two bits to give:

10.11

Which represents a value of two and three quarters.

Although floating point numbers give considerable benefits in terms of their flexibility, arithmetic operations involving floating point numbers are inherently slower than corresponding operations on fixed point numbers. Therefore, where possible, the benefits of speed associated with fixed point numbers should be exploited.

When considering the implementation of an algorithm in hardware, its efficiency is a prime concern. Many algorithms for the solution of linear equations involve computationally expensive division and/or multiplication operations. These operations should, where possible be avoided, although this is often not possible with known methods for solving linear equations.

Many systems of linear equations have sparse solutions. In such cases the number of iterations required to solve the system of equations should be relatively low. However, this does not occur with some known methods.

In summary there is a need for a more efficient algorithm which can be used to solve systems of linear equations.

It is an object of the present invention to obviate or mitigate at least one of the problems identified above.

Summary of the Invention

According to a first aspect of the present invention, there is provided a method for solving a system of N linear equations in N unknown variables, the method comprising:

(a) storing an estimate value for each unknown variable;

(b) initialising each estimate value to a predetermined value;

(c) for each estimate value:

(i) determining whether a respective predetermined condition is satisfied; and

(ii) updating the estimate if and only if the respective predetermined condition is satisfied; and

(d) repeating step (c) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

Thus, the invention provides a method for solving linear equations in which estimates for solutions of the equations are updated only if a predetermined condition is satisfied. The predetermined condition is preferably related to convergence of the method. Therefore such an approach offers considerable benefits in terms of efficiency, given that updates are only carried out when such updates are likely to accelerate convergence.

According to a second aspect of the present invention, there is provided a method for solving a system of N linear equations in N unknown variables, the method comprising:

(a) storing an estimate value for each unknown variable;

(b) initialising each estimate value to a predetermined value;

(c) attempting to update each estimate value using a scalar value d;

(d) updating the scalar value if no updates are made in step (c); and

(e) repeating step (c) and step (d) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

By updating the scalar value in accordance with the second aspect of the present invention, it has been discovered that benefits of efficiency are obtained.

According to a third aspect of the present invention, there is provided a method for solving a system of N linear equations in N unknown variables of the form:

$$\mathbf{Rh} = \beta$$

the method comprising:

generating a quadratic function of the form:

$$J(h) = \sum_{m=1}^{N}\sum_{n=1}^{N}\mathbf{R}(m,n)\mathbf{h}(m)\mathbf{h}(n) - 2\sum_{n=1}^{N}\beta(n)\mathbf{h}(n)\text{; and}$$

minimising said function using co-ordinate descent optimisation;

wherein **R** is a coefficient matrix of the system of linear equations; **h** is a vector of the N unknown variables; β is a vector containing the value of the right hand side of each equation; $\mathbf{R}(m,n)$ is an element of the matrix **R**; $\mathbf{h}(m)$ is the mth element of the matrix **h;** and $\beta(n)$ is the nth element of the vector β.

The present inventors have discovered that solving a system of linear equations by minimising a quadratic function using co-ordinate descent optimisation offers considerable and surprising efficiency benefits.

According to a fourth aspect of the present invention, there is provided a computer processor configured to solve a system of N linear equations in N unknown variables, comprising:

storage means for storing an estimate value for each unknown variable;

storage means for storing coefficients of each unknown variable in each equation;

storage means for storing a scalar value *d*;

initialising means for initialising each estimate value;

computing means configured to process each estimate value by determining whether a respective predetermined condition is satisfied, and to update the estimate if and only if the respective predetermined condition is satisfied, said computing means being configured to repeatedly process each estimate value until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

According to a fifth aspect of the present invention, there is provided a computer processor configured to solve a system of N linear equations in N unknown variables, comprising:

storage means for storing an estimate value for each unknown variable;

storage means for storing coefficients of each unknown variable in each equation;

storage means for storing a scalar value *d*;

initialising means for initialising each estimate value;

computing means configured to:

(a) attempt to update each estimate value using a scalar value *d*,

(b) update the scalar value *d* if no updates are made in step (a); and

(c) repeat step (a) and step (b) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

According to a sixth aspect of the present invention, there is provided a multiuser receiver device for obtaining data transmitted by a plurality users, the device comprising:

a plurality of filters, each filter being arranged to filter out a spreading code used by a respective user;

equation solving means to find a solution **h** of a system of linear equations of the form $\mathbf{Rh} = \beta$ where R is the cross correlation of the spreading codes used by the plurality of users, and β is a vector containing the filter output signals; and

means for obtaining the transmitted data using a solution provided by the equation solving means;

wherein the equation solving means:

(a) stores an estimate value for each value of the solution **h** ;

(b) initialises each estimate value to a predetermined value;

(c) for each estimate value:

(i) determines whether a respective predetermined condition is satisfied; and

(ii) updates the estimate if and only if the respective predetermined condition is satisfied; and

(d) repeats step (c) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

According to a seventh aspect of the present invention, there is provided a multiuser receiver device for obtaining data transmitted by a plurality of users, the device comprising:

a plurality of filters, each filter being arranged to filter out a spreading code used by a respective user;

equation solving means to find a solution **h** of a system of linear equations of the form $\mathbf{Rh} = \beta$ where R is the cross correlation of the spreading codes used by the plurality of users, and β is a vector containing the filter output signals; and

means to obtain the transmitted data using a solution provided by the equation solving means;

wherein the equation solving means:

(a) stores an estimate value for each unknown variable;

(b) initialises each estimate value to a predetermined value;

(c) attempts to update each estimate value using a scalar value d;

(d) updates the scalar value if no updates are made in step (c); and

(e) repeats step (c) and step (d) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

According to an eighth aspect of the present invention, there is provided a method for generating filter coefficients for use in an echo cancellation apparatus, the method comprising:

(a) generating a cross correlation matrix **R** containing the cross correlation of first and second signals and ;

(b) generating an auto correlation vector β containing an autocorrelation of the first signal; and

(c) determining a vector **h** for which $\mathbf{Rh} = \beta$, said vector **h** containing the said filter coefficients;

wherein the vector **h** is determined by solving the system of equations $\mathbf{Rh} = \beta$ by:

(d) storing an estimate value for each element of the vector **h**;

(e) initialising each estimate value to a predetermined value;

(f) for each estimate value:

(i) determining whether a respective predetermined condition is satisfied; and

(ii) updating the estimate if and only if the respective predetermined condition is satisfied; and

(g) repeating step (f) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

According to a ninth aspect of the present invention, there is provided a method for generating filter coefficients for use in an echo cancellation apparatus, the method comprising:

(a) generating a cross correlation matrix **R** containing the cross correlation of first and second signals;

(b) generating an auto correlation vector β containing an autocorrelation of the first signal; and

(c) determining a vector **h** for which $\mathbf{Rh} = \beta$, said vector **h** containing the said filter coefficients;

wherein the vector **h** is determined by solving the system of equations $\mathbf{Rh} = \beta$ by:

(d) storing an estimate value for each unknown variable;

(e) initialising each estimate value to a predetermined value;

(f) attempting to update each estimate value using a scalar value *d*;

(g) updating the scalar value if no updates are made in step (f); and

(h) repeating step (g) and step (h) until each estimate value is sufficiently close to an accurate value of the respective unknown variable.

Brief description of drawings

Embodiments of the present invention will now be described, by way of example, with reference to the accompanying drawings, in which:

Figure 1 is a flow chart of an algorithm for solving linear equations in accordance with the present invention;

Figure 2 is a graph showing how the value of the solution vector changes as the algorithm of Figure 1 solves a system of equations;

Figure 3 is a graph showing how the error of the solution vector varies as the algorithm of Figure 1 solves the system of equations;

Figure 4 is a graph showing how the number of passes through the system equations varies in dependence upon the bit number of the solution vector elements being considered in the algorithm of Figure 1;

Figure 5 is a graph showing the number of updates to the solution vector carried out for each bit as the algorithm of Figure 1 solves the system of equations;

Figure 6 is a flow chart showing a variant to the algorithm of Figure 1;

Figure 7 is a MATLAB code fragment implementing the algorithm of Figure 6;

Figure 8 is a flow chart showing a variant to the algorithm of Figure 6, where values of the solution vector are constrained between upper and lower bounds.

Figure 9 is flow chart showing how the algorithm of Figure 1 can be adapted to solve equations having complex coefficients and complex solutions;

Figure 10 is a flow chart showing a method for determining the next course of action in a step of Figure 9;

Figure 11 is a flow chart showing a variant to the algorithm of Figure 10;

Figure 12 is a MATLAB code fragment implementing the algorithm of Figure 11;

Figure 13 is a schematic illustration of a device configured to solve linear equations in accordance with the invention;

Figure 14 is a schematic illustration of the equation solving microprocessor of Figure 13;

Figure 15 is a schematic illustration showing block **R** of Figure 14 in further detail;

Figure 16 is a schematic illustration showing block **h** of Figure 14 in further detail;

Figure 17 is a schematic illustration showing how the block **h** of Figure 16 is modified when equations having complex valued solutions are to be solved;

Figure 18 is a schematic illustration showing block **Q** of Figure 14 in further detail;

Figure 19 is a schematic illustration showing the minimisation block of Figure 14 in further detail;

Figure 19a is a MATLAB code fragment showing a variant to the algorithm of Figure 7;

Figure 20 is a schematic illustration of a CDMA receiver device in which algorithms of the present invention may be applied;

Figure 21 is a schematic illustration of an echo cancellation apparatus in which the algorithms of the present invention may be applied; and

Figure 22 is a schematic illustration showing part of the apparatus of Figure 21 in further detail.

Description of preferred embodiments

A method for a system of solving linear equations is now described. A system of linear equations can be expressed in the form:

$$\mathbf{Rh} = \beta \tag{1}$$

where: **R** is a coefficient matrix of the system of equations;

h is a vector of the unknown variables; and

β is a vector containing the value of the right hand side of each equations

For example, the system of equations (2):

$$\begin{aligned} 15x + 5y - 2z &= 15 \\ 5x + 11y + 4z &= 47 \\ -2x + 4y + 9z &= 51 \end{aligned} \qquad (2)$$

can be expressed in the form of equation (1) where:

$$\mathbf{R} = \begin{bmatrix} 15 & 5 & -2 \\ 5 & 11 & 4 \\ -2 & 4 & 9 \end{bmatrix} \qquad \mathbf{h} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \beta = \begin{bmatrix} 15 \\ 47 \\ 51 \end{bmatrix} \qquad (3)$$

To solve the system of equations, it is necessary to find values for x, y, and z of **h** which satisfy each of the three equations.

In operation, algorithm uses the matrix **R** and the vectors **h** and β as set out above, together with an auxiliary vector **Q.** The vector **h** is initialised to a predetermined initial value (see below) and updated as the algorithm proceeds until its elements represent the solution of the equations.

For a system of N equations in N unknown variables, the vector **h** has length N and the matrix **R** is of size N x N.

Referring to Figure 1, at step S1 the vectors **h** and **Q** are initialised. The vector **h** is initialised such that all its elements are set to '0'. The vector **Q** is initialised to contain the negative of the equivalent position of β. That is:

$$\mathbf{Q} = -\beta \qquad (4)$$

Therefore, when working with system of equations (2), **Q** is initialised in accordance with equation (5):

$$\mathbf{Q} = \begin{bmatrix} -15 \\ -47 \\ -51 \end{bmatrix} \tag{5}$$

The algorithm maintains three counter variables *p*, *m* and *it*, a parameter *N* which represents the number of elements in the solution vector (and also the number of equations), a parameter *M* which represents the number of bits used to represent each element of the solution vector **h**, a parameter *Nit* which represents the maximum number of iterations through which the algorithm can pass for a particular value of *m*, a variable *Flag* which is used to indicate whether or not the solution vector has been updated, and a constant *H*, the purpose of which is described below.

Some of these variables are initialised at step S2 and step S3 of Figure 1. *p*, *m* and *it* are all initialised to zero. *N, M, and Nit* are set to the values described above which can either be chosen by a user or hard coded into the algorithm. Selection and initialisation of *H* is described below.

Operation of the algorithm can be summarised as follows. Each bit *m* of all elements *p* of the solution vector **h** is considered in turn. As described below, for each bit an element of the vector **Q** is compared with various conditions and the result of this comparison determines whether or not further processing is applicable. This further processing comprises an appropriate update of the element *p* of the solution vector **h** corresponding to the element being considered and updates of all elements of the auxiliary vector **Q**.

When it is determined that further processing for that element is not appropriate (for the current bit), the next element is considered. When each element has been considered for that particular bit, all elements of the solution vector are considered for the next bit in turn, and updated appropriately. This process continues until all elements have been considered for all bits. If the total number of iterations for any one bit reaches a predetermined limited the algorithm again ends. The algorithm is described in further detail below.

At step S3, the value of m is incremented to '1'. Thus, the algorithm is now considering the first bit of each element in the solution vector **h**. *it* is set to 0 to indicate that no iterations have yet taken place for the current value of m.

At Step S4, a step size parameter d is calculated according to the equation:

$$d = 2^{-m} H \tag{6}$$

where m and H are the parameters described above.

H is a value greater than or equal to the magnitude of the maximum value which is expected for any value of the solution vector. That is, the algorithm considers only solutions lying between $-H$ and $+H$.

As will be described below, setting d in accordance with equation (6) allows each bit of each value of the solution vector **h** to be considered in turn.

At Step S5 of Figure 1, the variable *it* is incremented, and the variable *Flag* is set to '0'. p (the current element of the solution vector under consideration) is incremented at Step S6.

Having performed the necessary incrementation and initialisation, the algorithm can begin processing elements of the matrix and vectors, in an attempt to solve the equations.

At step S7, the following operation is performed:

$$\text{arg} = \arg\min\left\{\mathbf{Q}(p), -\mathbf{Q}(p), -\mathbf{R}(p,p)\cdot\frac{d}{2}\right\} \tag{7}$$

where

$$\arg\min = \begin{cases} 1, \text{if} \left\{ \mathbf{Q}(p) < -\mathbf{Q}(p) \wedge \mathbf{Q}(p) < -\mathbf{R}(p,p) \cdot \frac{d}{2} \right\} \\ 2, \text{if} \left\{ -\mathbf{Q}(p) < \mathbf{Q}(p) \wedge -\mathbf{Q}(p) < -\mathbf{R}(p,p) \cdot \frac{d}{2} \right\} \\ 3, \text{if} \left\{ -\mathbf{R}(p,p) \cdot \frac{d}{2} \leq \mathbf{Q}(p) \wedge -\mathbf{R}(p,p) \cdot \frac{d}{2} \leq -\mathbf{Q}(p) \right\} \end{cases} \quad (8)$$

The value of arg is assessed at the decision block of step S8.

If arg = 1, the element of the solution vector under consideration, that is $\mathbf{h}(p)$ is set according to equation (9) at step S9.

$$\mathbf{h}(p) = \mathbf{h}(p) + d \quad (9)$$

The auxiliary vector $\mathbf{Q}$ is then updated such that all values of $\mathbf{Q}$ are set according to equation (10) at step S10.

$$\mathbf{Q}(r) = \mathbf{Q}(r) + d\mathbf{R}(p,r), \forall r : 1 \leq r \leq N \quad (10)$$

If arg = 2, the element of the solution vector under consideration, that is $\mathbf{h}(p)$, is set according to equation (11) at step S11.

$$\mathbf{h}(p) = \mathbf{h}(p) - d \quad (11)$$

The auxiliary vector $\mathbf{Q}$ is then updated such that all values of $\mathbf{Q}$ are set according to equation (12) at step S10.

$$\mathbf{Q}(r) = \mathbf{Q}(r) - d\mathbf{R}(p,r), \forall r : 1 \leq r \leq N \quad (12)$$

If arg = 1 or if arg = 2, *Flag* is set to '1' at step S13.

If arg = 3, no update is made to any element of the solution vector **h** or the auxiliary vector **Q**, and *Flag* is not updated.

Having made the updates set out above, a decision block at step S14 checks the condition of equation (13);

$$p=N \tag{13}$$

If p is not equal to N (i.e. all elements of the solution vector **h** have not yet been considered), control returns to step S6 and p is incremented. This process continues until all entries in the solution vector **h** have been considered, and **h** and **Q** are updated in the manner set out above.

If p is equal to N (step S14), a check is made to determine the value of *Flag* (step S15).

Flag is initially set to '0' at step S5, and only updated (to be equal to '1') if entries of the solution and auxiliary vectors are amended by steps S9 and S10, or steps S11 and S12. Thus, if Flag = 1, it can be deduced that a change was made to at least one element of **h** (i.e. one **h**(p) value) and all values of **Q**, for the current iteration *it*. Therefore, assuming that the total number of iterations *it* has not exceeded the limit set by N*it* (checked at S16), p is reset to '0' at step S17, control returns to step S5, and the current bit is again processed for each element p of the solution vector **h**. This is because further processing of each element of **h** for the current value of m may result in further updates. If the total number of iterations has reached the limit set by N*it* (step S16), the algorithm exits.

If it is the case that Flag = 0 (step 15), it can be deduced that no updates have been made to any elements of the solution vector **h** or the auxiliary vector **Q** for any value of p (that is any element of the solution vector **h**). Given that further iterations of steps S5 to S15 will result in no changes to the elements of **h** (given that neither *d*, **h** nor **Q** have changed), a check is made to determine whether or not all bits m have

been considered (step S18), by comparing the current value of m with the total number of bits M. If it is the case that $m=M$, i.e. all bits have been considered, there is no work for the algorithm to do, and the algorithm exits at step S19.

If it is the case that all bits have not been considered, p is reset to 0 at step S20, and control returns to step S3, and the algorithm processes the next bit of the solution vector entries.

In the preceding discussion, it has been explained that entries of the solution vector **h** are processed for each bit of the solution vector entries, starting with the most significant bit. However, it can be seen from the preceding discussion, that at all steps the entire value of an element of **h** is used for update. However bit wise processing is in fact achieved because following each increment of m (step S3) a new value of d is created at step S4. Given that each increment of m will result in the value of d being divided by two (given the presence of the expression 2^{-m} in the expression for d), and given that d is used to update both **h** and **Q**, after an update of d the next most significant bit is then updated.

It has been described that the value of H represents a value greater than or equal to the magnitude of the maximum value of the solution vector elements. In setting H it is desirable to ensure that it is a power of two. That is, H is set according to equation (14).

$$H = 2^q \tag{14}$$

where q is any integer (i.e. positive, negative or zero)

When H is set in this way, the expression for d set out in equation (6) becomes:

$$d = 2^{-m}2^q$$
$$d = 2^{q-m} \tag{15}$$

Thus, when H is chosen in accordance with equation (14), the value of d can be updated without multiplication or division, simply by appropriate bit shift operations. This is particularly advantageous, because microprocessors can typically carry out bit shift operations far more efficiently than multiplication or division.

The application of the algorithm described with reference to Figure 1 to the system of linear equations (2) set out above will now be described.

R and β are initialised as described above:

$$\mathbf{R} = \begin{bmatrix} 15 & 5 & -2 \\ 5 & 11 & 4 \\ -2 & 4 & 9 \end{bmatrix} \tag{16}$$

$$\beta = \begin{bmatrix} 15 \\ 47 \\ 51 \end{bmatrix} \tag{17}$$

h and **Q** are initialised in the manner described above:

$$\mathbf{h} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{18}$$

$$\mathbf{Q} = \begin{bmatrix} -15 \\ -47 \\ -51 \end{bmatrix} \tag{19}$$

Variables are initialised as follows at step S2 and step S3:

$$\begin{aligned} m &= 0 & M &= 8 \\ it &= 0 & N &= 3 \\ p &= 0 & \mathrm{N}it &= 0 \end{aligned} \tag{20}$$

H is in this case set to 16, i.e. $q = 4$ in equation (14).

m is incremented such that m=1, and it is set to '0' (step S3). A value of d is computed according to equation (15) and in this case:

$$\begin{aligned} d &= 2^{4-1} \\ d &= 8 \end{aligned} \tag{21}$$

it is incremented to '1' and Flag is set to '0' at step S5. p is incremented to 1 at step S6. At step S7, the following expression is evaluated

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(1), -\mathbf{Q}(1), -\mathbf{R}(1,1)\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\left\{-15, 15, -15\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\{-15, 15, -60\} \end{aligned} \tag{22}$$

Therefore, it can be seen from equations (8) and (22) that arg =3. The decision block at step S8 therefore passes control to step S14, where the condition p=N is checked. In this case p=1 and N =3, therefore p is not equal to N, and therefore control passes to step S6, with no changes having been made to the elements of **h** or **Q**.

Step S6 then increments p to be equal to 2 (i.e. the second element of the solution vector is being considered)

Step S7 computes:

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(2), -\mathbf{Q}(2), -\mathbf{R}(2,2)\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\left\{-47, 47, -11\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\{-47, 47, -44\} \end{aligned} \tag{23}$$

Therefore, arg=1.

At step S8 the appropriate decision outcome is chosen, and at step S9, **h** is set as follows:

$$\mathbf{h}:\begin{bmatrix}0\\0\\0\end{bmatrix}\xrightarrow{\text{becomes}}\begin{bmatrix}0\\0+d\\0\end{bmatrix}=\begin{bmatrix}0\\8\\0\end{bmatrix} \tag{24}$$

At step S10, all values of **Q** are set by adding the current values to the values of the second row (since p=2) of **R** multiplied by *d*:

$$\mathbf{Q}:\begin{bmatrix}-15\\-47\\-51\end{bmatrix}\xrightarrow{\text{becomes}}\begin{bmatrix}-15+d\cdot\mathbf{R}(2,1)\\-47+d\cdot\mathbf{R}(2,2)\\-51+d\cdot\mathbf{R}(2,3)\end{bmatrix}=\begin{bmatrix}-15+8\cdot5\\-47+8\cdot11\\-51+8\cdot4\end{bmatrix}=\begin{bmatrix}25\\41\\-19\end{bmatrix} \tag{25}$$

At step S13, Flag is set to '1' to show that **h** and **Q** have been updated.

At step S14 *p* is still not equal to *N* (*p*=2, *N*=3) and therefore control returns to step S6, where *p* is incremented to 3.

At step S7 the following is computed:

$$\begin{aligned}
\arg &= \arg\min\left\{\mathbf{Q}(3),-\mathbf{Q}(3),-\mathbf{R}(3,3)\cdot\frac{8}{2}\right\} \\
\arg &= \arg\min\left\{-19,19,-9\cdot\frac{8}{2}\right\} \\
\arg &= \arg\min\{-19,19,-36\}
\end{aligned} \tag{26}$$

Therefore arg=3. No updates are made, and the condition of step S14 is checked. In this instance $p=N=3$, and the condition is therefore true. The value of *Flag* is then checked at step S15. *Flag* was set to '1' at step S13 while p was set to 2, and therefore the condition of step S15 evaluates to false.

The condition of step S16 is then checked. Given that $it=1$, and $Nit=10$, the condition of step S16 returns False, and control passes to step S17 where p is reset to '0' and then to step S5, where *it* is incremented and *Flag* is reset to '0'.

p is incremented at step S6, and at step S7, the following expression is considered

$$\begin{aligned}
\arg &= \arg\min\left\{\mathbf{Q}(1),-\mathbf{Q}(1),-\mathbf{R}(1,1)\cdot\frac{8}{2}\right\} \\
\arg &= \arg\min\left\{25,-25,-15\cdot\frac{8}{2}\right\} \\
\arg &= \arg\min\{25,-25,-60\}
\end{aligned} \tag{27}$$

It can be seen from equation (27) that arg =3. The decision block therefore passes control to step S14, where the condition $p=N$ is checked. In this case $p=1$ and $N=3$, therefore p is not equal to N, and control therefore passes to step S6, with no changes having been made to the elements of $\mathbf{h}$ or $\mathbf{Q}$.

p is incremented to 2 at step S6 and the following expression is considered at step S7:

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(2),-\mathbf{Q}(2),-\mathbf{R}(2,2)\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\left\{41,-41,-11\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\{41,-41,-44\} \end{aligned} \tag{28}$$

It can be seen from equation (28) that arg =3. The decision block therefore passes control to step S14, where the condition $p=N$ is again checked. p is still not equal to N, and therefore control passes to step S6, with no changes having been made to the elements of **h** or **Q**.

The following expression is then considered:

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(3),-\mathbf{Q}(3),-\mathbf{R}(3,3)\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\left\{-19,19,-9\cdot\frac{8}{2}\right\} \\ \arg &= \arg\min\{-19,19,-36\} \end{aligned} \tag{29}$$

Again, arg=3, and no updates are made. In this case the condition of step S14 returns true, and the condition of step S15 also returns true, given that no updates where made during the last pass through all elements of **h**, and consequently *Flag* is still set to '0'.

The condition of step S18 is then checked to determine whether all bits of the solution vector entries have been considered. In this case $m = 1$ and $M = 8$, therefore step S18 returns false. *p is* set to '0' at step S20, and control passes to step S3 where m is incremented to 2 and *it* is reset to '0'.

At step S4 d is set where m is equal to 2, and therefore d=4. Note that as discussed above, d has been halved. At step S5 *it* is incremented to 1 and Flag is set to '0'.

p is incremented to 1 at step S6. At step S7, the following expression is considered:

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(1),-\mathbf{Q}(1),-\mathbf{R}(1,1)\cdot\frac{4}{2}\right\} \\ \arg &= \arg\min\left\{25,-25,-15\cdot\frac{4}{2}\right\} \\ \arg &= \arg\min\{25,-25,-30\} \end{aligned} \tag{30}$$

Therefore arg=3, and no update takes place. The algorithm continues as described above, and p is incremented to 2 at step S6. At step S7 the following expression is evaluated:

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(2),-\mathbf{Q}(2),-\mathbf{R}(2,2)\cdot\frac{4}{2}\right\} \\ \arg &= \arg\min\left\{41,-41,-11\cdot\frac{4}{2}\right\} \\ \arg &= \arg\min\{41,-41,-22\} \end{aligned} \tag{31}$$

In this case arg=2. The decision block of step S8 therefore directs control to step S11. Step S11 updates **h** by subtracting the current value of d (d=4) from the second element of **h**, as shown in equation (32):

$$\mathbf{h}:\begin{bmatrix}0\\8\\0\end{bmatrix}\xrightarrow{\text{becomes}}\begin{bmatrix}0\\8-d\\0\end{bmatrix}=\begin{bmatrix}0\\4\\0\end{bmatrix} \tag{32}$$

Step S12 updates **Q** by subtracting the row of **R** (given that p=2) multiplied by d from the current values of **Q** as set out in equation (33):

$$\mathbf{Q}:\begin{bmatrix}25\\41\\-19\end{bmatrix}\xrightarrow{\text{becomes}}\begin{bmatrix}25-d\cdot\mathbf{R}(2,1)\\41-d\cdot\mathbf{R}(2,2)\\-19-d\cdot\mathbf{R}(2,3)\end{bmatrix}=\begin{bmatrix}25-4\cdot5\\41-4\cdot11\\-19-4\cdot4\end{bmatrix}=\begin{bmatrix}5\\-3\\-35\end{bmatrix} \tag{33}$$

Flag is set to 1 at step S13.

Another iteration is then carried out, wherein p is set to 3 at step S6, and the following expression is considered at step S7:

$$\begin{aligned} \arg &= \arg\min\left\{\mathbf{Q}(3), -\mathbf{Q}(3), -\mathbf{R}(3,3)\cdot\frac{4}{2}\right\} \\ \arg &= \arg\min\left\{-35, 35, -9\cdot\frac{4}{2}\right\} \\ \arg &= \arg\min\{-35, 35, -18\} \end{aligned} \tag{34}$$

Therefore, arg is set to 1, the decision block of step S8 directs control to step S9, and steps S9 and S10 set **h** and **Q** as set out below:

$$\mathbf{h}: \begin{bmatrix} 0 \\ 4 \\ 0 \end{bmatrix} \xrightarrow{\text{becomes}} \begin{bmatrix} 0 \\ 4 \\ 0+d \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 4 \end{bmatrix} \tag{35}$$

$$\mathbf{Q}: \begin{bmatrix} 5 \\ -3 \\ -35 \end{bmatrix} \xrightarrow{\text{becomes}} \begin{bmatrix} 5+d\cdot\mathbf{R}(3,1) \\ -3+d\cdot\mathbf{R}(3,2) \\ -35+d\cdot\mathbf{R}(3,3) \end{bmatrix} = \begin{bmatrix} 5+4\cdot-2 \\ -3+4\cdot 4 \\ -35+4\cdot 9 \end{bmatrix} = \begin{bmatrix} -3 \\ 13 \\ 1 \end{bmatrix} \tag{36}$$

Given that p=N and *Flag*=1, p is reset (step S17) and control passes to S5. For each of the three values of p steps S6 to S14 are executed. On each occasion, arg=3 and no updates take place. The individual expressions considered by step S7 during each iteration are not set out in full, although they can be readily deduced from the information presented above.

Given that Flag=0, the algorithm continues for the next value of m.

Execution of the algorithm then continues in the manner outlined above, for each bit of the solution vector elements in turn.

The values of **h** after each iteration through the solution vector elements are set out below. It should be noted that iteration number referred to here is equivalent to a cumulative iteration count instead of the bit by bit iteration count *it* described above.

Iteration :	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1st element :	0	0	0	0	0	0	0	1	1	1	1
2nd element :	0	8	8	4	4	2	2	2	2	2	2
3rd element :	0	0	0	4	4	4	4	5	5	5	5

Figure 2 is a graph showing the value of each solution vector element after each iteration. A first line 1 represents changes to the first solution vector element, **h**(1), a second line 2 represents changes to the second solution vector element **h**(2), and a third line 3 represents changes to the third solution vector element **h**(3).

Solving the set of equations set out at (2) above in a conventional way yields

$$\mathbf{h} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} \tag{37}$$

Thus, it can be seen that the algorithm effectively solves the system of equations after seven passes through the solution vector elements.

The error in the values of **h** after each iteration is shown below:

Iteration :	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1st element :	-1	−1	−1	−1	−1	−1	−1	0	0	0	0
2nd element :	-2	6	6	2	2	0	0	0	0	0	0
3rd element :	-5	−5	−5	−1	−1	−1	−1	0	0	0	0

These values are plotted in the graph of Figure 3, where a first line 4 represents changes to the error of the first solution vector element, **h**(1), a second line 5 represents changes to the error of the second solution vector element **h**(2), and a third line 6 represents changes to the error of the third solution vector element **h**(3). It can be seen that errors diminish as the algorithm proceeds.

Figure 4 is a graph showing the number of iterations carried out for each bit, i.e. the value of *it* when processing of each bit *m* has been completed. Figure 5 shows the number of updates made to the solution vector **h** and auxiliary vector **Q** for each bit *m*.

As described above, the auxiliary vector **Q** is updated as the algorithm progresses. The values of **Q** after each update of the vector **Q** and the solution vector **h** are set out below:

$$\begin{bmatrix} \text{Update} & \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \text{1st element} & -15 & 25 & 5 & -3 & -13 & 2 & 0 \\ \text{2nd element} & -47 & 41 & -3 & 13 & -9 & -4 & 0 \\ \text{3rd element} & -51 & -19 & -35 & 1 & -7 & -9 & 0 \end{bmatrix}$$

As a further example, consider the system of equations set out below:

$$\begin{aligned} 15x + 5y - 2z &= 15 \\ 5x + 11y + 4z &= -37 \\ -2x + 4y + 9z &= -55 \end{aligned} \tag{38}$$

In this case the accurate solution of the equations is:

$$\mathbf{h} = \begin{bmatrix} 1 \\ -2 \\ -5 \end{bmatrix} \tag{39}$$

It should also be noted that the value H is now equal to 256. Other parameters of the algorithm remain unchanged.

The value of the solution vector after each pass of the algorithm is set out below. Again, it can be seen that the algorithm correctly solves the system of equations.

$$\begin{bmatrix} \textbf{Iteration:} & \textbf{0} & \textbf{1} & \textbf{2} & \textbf{3} & \textbf{4} & \textbf{5} & \textbf{6} & \textbf{7} & \textbf{8} & \textbf{9} & \textbf{10} & \textbf{11} & \textbf{12} \\ \text{1st element:} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \text{2nd element:} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & -2 & -2 \\ \text{3rd element:} & 0 & 0 & 0 & 0 & 0 & -8 & -8 & -8 & -6 & -6 & -6 & -5 & -5 \end{bmatrix} \tag{40}$$

Figure 6 shows a variant to the algorithm described above with reference to Figure 1. Many steps of Figure 6 are identical to steps of Figure 1. Such steps are identified by like reference numerals in both Figure 1 and Figure 6. Only steps which differ from Figure 1 are described in further detail below.

In Figure 6, Step S4 of Figure 1 is replaced by step S21. It can be seen that in addition to setting d in accordance with equation (6):

$$d = 2^{-m} H \tag{6}$$

a two element array *delta* is established as follows:

$$delta[1] = d \tag{41}$$

$$delta[2] = -d \tag{42}$$

In Figure 1 step S8 determines the value of arg (i.e. 1, 2 or 3) and chooses a different action in dependence upon the value. In Figure 6, step S8 is replaced with a single comparison at step S22:

$$\text{arg} < 3 \tag{43}$$

If the condition returns false, it can be determined that arg=3 (given that arg can only ever take values of '1', '2', and '3'). Therefore in accordance with the algorithm of Figure 1, Figure 6 shows that if the condition is false, no updates to **h** or **Q** are made, and control passes to step S14 as in Figure 1.

If the condition of step S22 of Figure 6 is satisfied, it can be deduced that arg = 1 or arg = 2. In Figure 1, if arg = 1, steps S9 and S10 update **h** and **Q** using expressions including d. Similarly, if arg = 2, steps S11 and S12 update **h** and **Q** using expressions including $-d$.

In the variant of the algorithm shown in Figure 6, the steps S9 and S11 of Figure 1 are replaced with a single step S23, and similarly steps S10 and S12 are replaced with a single step S24. Both of steps S23 and S24 involve the array *delta* and more particularly the element arg of the array *delta*.

Step S23 updates **h** according to equation (44) and step S24 updates **Q** according to equation (45).

$$\mathbf{h}(p) = \mathbf{h}(p) + delta(\arg) \tag{44}$$

$$\mathbf{Q}(r) = \mathbf{Q}(r) + delta(\arg) \cdot \mathbf{R}(p,r), \forall r : r = 1,\ldots,N \tag{45}$$

Given that the first element of *delta* contains d and the second element contains $-d$, it can be seen that equations (44) and (45) correctly correspond to equivalent operations of the algorithm of Figure 1.

Figure 7 shows MATLAB® source code for implementing the algorithm illustrated in Figure 6.

Figure 8 is a flow chart showing a further variant to the algorithm described above. The algorithm of Figure 8 is intended to be used where upper and lower bounds are

to be enforced upon elements of the solution vector **h**. For example, it may be known that the solutions lie between +5 and -5, and it may therefore speed up execution of the algorithm if only values in this range are considered.

Figure 8 illustrates an algorithm where all values **h**(p) of the solution vector **h** are constrained with the bounds of equation (46):

$$h_1 < \mathbf{h}(p) < h_2 \tag{46}$$

Before execution of the algorithm constants h_1 and h_2 are established. The establishment of these constants is not illustrated in the flowchart of Figure 8. Steps of Figure 8 which are identical to steps of Figure 6 are identified by like reference numerals in both figures. Only steps which are different between the two Figures are described in further detail below.

Having updated **h**(p) at step S23, a test is executed at a decision block of step S25 to ensure that newly set value of **h**(p) lies between the bounds specified in equation (46). If the new value of **h**(p) satisfies equation (46), the algorithm proceeds to update the auxiliary vector **Q** at S24 and to update *Flag* at step S13 as described above with reference to Figure 6.

If step S25 determines that the newly set value of **h**(p) does not satisfy equation (46), **h**(p) is again updated at step S26. This updating reverses the update of step S23, such that **h**(p) is equal to the value before execution of step S23. Therefore, **h**(p) has not been updated (because the attempted update did not satisfy equation (46)), Flag is not set to '1', and execution of the algorithm continues at step S14.

In yet a further embodiment of the present invention, the constants h_1 and h_2 of the embodiment of Figure 8 are replaced by vectors $\mathbf{h}_1$ and $\mathbf{h}_2$ both of which have a length equal to the solution vector **h**. The vector $\mathbf{h}_1$ contains a lower bound for each

element of the solution vector, and the vector $\mathbf{h_2}$ contains an upper bound for each element of the solution vector. Thus, the condition of step S25 becomes:

$$\mathbf{h}_1(p) < \mathbf{h}(p) < \mathbf{h}_2(p) \tag{46}$$

The embodiments of the present invention described above are concerned with the application of the algorithm to systems of equations which have real valued solutions. The present invention is also applicable to the solution of systems of equations having complex valued solutions, and the application of the invention to such systems of equations is described below.

Consider the system of equations set out below:

$$\begin{aligned}(a_1 + a_2 i)x + (b_1 + b_2 i)y + (c_1 + c_2 i)z &= A_1 + A_2 i \\ (d_1 + d_2 i)x + (e_1 + e_2 i)y + (f_1 + f_2 i)z &= B_1 + B_2 i \\ (g_1 + g_2 i)x + (h_1 + h_2 i)y + (j_1 + j_2 i)z &= C_1 + C_2 i\end{aligned} \tag{47}$$

where each of the unknown variables x, y and z is a complex number defined as follows:

$$\begin{aligned}x &= x_1 + x_2 i \\ y &= y_1 + y_2 i \\ z &= z_1 + z_2 i\end{aligned} \tag{48}$$

and

$$i = \sqrt{-1}$$

From equation (1) above, the system of equations (47) can be expressed as follows:

$$\mathbf{R} = \begin{bmatrix} a_1 + a_2 i & b_1 + b_2 i & c_1 + c_2 i \\ d_1 + d_2 i & e_1 + e_2 i & f_1 + f_2 i \\ g_1 + g_2 i & h_1 + h_2 i & j_1 + j_2 i \end{bmatrix} \tag{49}$$

$$\mathbf{h} = \begin{bmatrix} x_1 + x_2 i \\ y_1 + y_2 i \\ z_1 + z_2 i \end{bmatrix} \tag{50}$$

$$\beta = \begin{bmatrix} A_1 + A_2 i \\ B_1 + B_2 i \\ C_1 + C_2 i \end{bmatrix} \tag{51}$$

To solve the system of equations, a matrix **A**, and vectors **b**, and **c** are created from the data set out above. **A** is a $2N$ by $2N$ real-valued coefficient matrix, **b** is a real-valued solution vector of length $2N$, and **c** is a real-valued right hand side vector of length $2N$, where N is the number of unknown variables (i.e. N=3 in this example).

A, **b**, and **c** are set as described below:

$$\begin{aligned} \mathbf{A}(2m-1,2n-1) &= \mathrm{Re}\{\mathbf{R}(m,n)\} \\ \mathbf{A}(2m,2n) &= \mathrm{Re}\{\mathbf{R}(m,n)\} \\ \mathbf{A}(2m-1,2n) &= \mathrm{Im}\{\mathbf{R}(m,n)\} \\ \mathbf{A}(2m,2n-1) &= -\mathrm{Im}\{\mathbf{R}(m,n)\} \end{aligned} \tag{52}$$

$$\begin{aligned} \mathbf{b}(2n-1) &= \mathrm{Re}\{\mathbf{h}(n)\} \\ \mathbf{b}(2n) &= \mathrm{Im}\{\mathbf{h}(n)\} \end{aligned} \tag{53}$$

$$\begin{aligned} \mathbf{c}(2n-1) &= \mathrm{Re}\{\beta(n)\} \\ \mathbf{c}(2n) &= \mathrm{Im}\{\beta(n)\} \end{aligned} \tag{54}$$

Where Re{} is a function returning the real coefficient of a complex number, and Im{} is a function returning the imaginary coefficient of a complex number.

Thus, in the case of equations (47), **A**, is set as follows:

$$\mathbf{A}=\begin{bmatrix} a_1 & a_2 & b_1 & b_2 & c_1 & c_2 \\ -a_2 & a_1 & -b_2 & b_1 & -c_2 & c_1 \\ d_1 & d_2 & e_1 & e_2 & f_1 & f_2 \\ -d_2 & d_1 & -e_2 & e_1 & -f_2 & f_1 \\ g_1 & g_2 & h_1 & h_2 & j_1 & j_2 \\ -g_2 & g_1 & -h_2 & h_1 & -j_2 & j_1 \end{bmatrix} \qquad (55)$$

The present invention is often used to solve normal equations. Where normal equations are solved, the matrix **A** is such that:

$$\begin{aligned} b_1 &= d_1 & a_1 &> 0 \\ b_2 &= -d_2 & e_1 &> 0 \\ f_1 &= h_1 & j_1 &> 0 \\ f_2 &= -h_2 & a_2 &= 0 \\ c_1 &= g_1 & e_2 &= 0 \\ c_2 &= -g_2 & j_2 &= 0 \end{aligned} \qquad (55a)$$

The vectors **b** and **c** are set as follows:

$$\mathbf{b}=\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \\ z_1 \\ z_2 \end{bmatrix} \qquad (56) \qquad \mathbf{c}=\begin{bmatrix} A_1 \\ A_2 \\ B_1 \\ B_2 \\ C_1 \\ C_2 \end{bmatrix} \qquad (57)$$

Having created the vectors **a** and **b** and the matrix **A** set out above, the methods for solving real valued equations set out above with reference to Figures 1, 6 and 8 can be used where:

$$\mathbf{R} = \mathbf{A}$$
$$\mathbf{h} = \mathbf{b} \qquad (58)$$
$$\beta = \mathbf{c}$$

Values of the solution vector **h** set by the algorithm can then be used to determine both the real and imaginary components of the complex numbers x, y and z, and to create the complex valued solutions.

In alternative embodiments of the present invention, the algorithms described above are modified such that the algorithm operates directly on **R**, **h** and β as set out in equations (49) to (51) above. These modifications are now described with reference to the flow chart of Figure 9.

The algorithm used to solve equations involving complex numbers is based upon that of Figure 1, but steps S7 to S12 are replaced by the steps shown in Figure 9. Step S27 of Figure 9 replaces step S7 of Figure 1, step S28 of Figure 9 replaces step S8 of Figure 1, and steps S29 to S36 replace steps S9 to S12 of Figure 1. Steps S13 and S14 are identical to the eqivilent steps of Figure 1 and are shown in dotted lines. They are included in Figure 9 for the sake of clarity.

Referring to Figure 9, at step S27, the following expression is evaluated:

$$\arg = \arg\min\left\{\mathrm{Re}\{\mathbf{Q}(p)\}, -\mathrm{Re}\{\mathbf{Q}(p)\}, \mathrm{Im}\{\mathbf{Q}(p)\}, -\mathrm{Im}\{\mathbf{Q}(p)\}, -\mathbf{R}(p,p)\cdot\frac{d}{2}\right\} \qquad (59)$$

where

$$\arg\min = \begin{cases} 1, \text{if} \left\{ \min(\text{Re}\{\mathbf{Q}(p)\}, -\text{Re}\{\mathbf{Q}(p)\}, \text{Im}\{\mathbf{Q}(p)\}, -\text{Im}\{\mathbf{Q}(p)\}, -\mathbf{R}(p,p)\cdot\frac{d}{2}) = \text{Re}\{\mathbf{Q}(p)\} \right\} \\ 2, \text{if} \left\{ \min(\text{Re}\{\mathbf{Q}(p)\}, -\text{Re}\{\mathbf{Q}(p)\}, \text{Im}\{\mathbf{Q}(p)\}, -\text{Im}\{\mathbf{Q}(p)\}, -\mathbf{R}(p,p)\cdot\frac{d}{2}) = -\text{Re}\{\mathbf{Q}(p)\} \right\} \\ 3, \text{if} \left\{ \min(\text{Re}\{\mathbf{Q}(p)\}, -\text{Re}\{\mathbf{Q}(p)\}, \text{Im}\{\mathbf{Q}(p)\}, -\text{Im}\{\mathbf{Q}(p)\}, -\mathbf{R}(p,p)\cdot\frac{d}{2}) = \text{Im}\{\mathbf{Q}(p)\} \right\} \\ 4, \text{if} \left\{ \min(\text{Re}\{\mathbf{Q}(p)\}, -\text{Re}\{\mathbf{Q}(p)\}, \text{Im}\{\mathbf{Q}(p)\}, -\text{Im}\{\mathbf{Q}(p)\}, -\mathbf{R}(p,p)\cdot\frac{d}{2}) = -\text{Im}\{\mathbf{Q}(p)\} \right\} \\ 5, \text{if} \left\{ \min(\text{Re}\{\mathbf{Q}(p)\}, -\text{Re}\{\mathbf{Q}(p)\}, \text{Im}\{\mathbf{Q}(p)\}, -\text{Im}\{\mathbf{Q}(p)\}, -\mathbf{R}(p,p)\cdot\frac{d}{2}) = -\mathbf{R}(p,p)\cdot\frac{d}{2} \right\} \end{cases} \tag{60}$$

and Re{} and Im{} are as defined above.

The value of arg set at step S27 is checked at the decision block of S28. This determines the update (if any) to be made to the elements of the solution vector **h** and the auxiliary vector **Q**.

If arg is 1, then the element p of the solution vector **h** currently under consideration is updated as follows at step S29:

$$\mathbf{h}(p) = \mathbf{h}(p) + d \tag{61}$$

Equation (61) means that the real part of $\mathbf{h}(p)$ is increased by d.

All elements of the auxiliary vector **Q** are also updated, in accordance with equation (62) at step S30.

$$\mathbf{Q}(r) = \mathbf{Q}(r) + d\mathbf{R}(p,r), \forall r : 1 \le r \le N \tag{62}$$

If arg is 2, then the element p of the solution vector **h** currently under consideration is updated as follows at step S31:

$$\mathbf{h}(p) = \mathbf{h}(p) - d \tag{63}$$

Equation (64) means that the real part of $\mathbf{h}(p)$ is decreased by d.

All elements of the auxiliary vector $\mathbf{Q}$ are also updated, in accordance with equation (62) at step S32.

$$\mathbf{Q}(r) = \mathbf{Q}(r) - d\mathbf{R}(p,r), \forall r : 1 \le r \le N \tag{64}$$

If arg is 3, then the element p of the solution vector $\mathbf{h}$ currently under consideration is updated as follows at step S33:

$$\mathbf{h}(p) = \mathbf{h}(p) + i \cdot d \tag{65}$$

Equation (65) means that the imaginary part of $\mathbf{h}(p)$ is increased by d.

All elements of the auxiliary vector $\mathbf{Q}$ are also updated, in accordance with equation (66) at step S34.

$$\mathbf{Q}(r) = \mathbf{Q}(r) + d \cdot i \cdot \mathbf{R}(p,r), \forall r : 1 \le r \le N \tag{66}$$

If arg is 4, then the element p of the solution vector $\mathbf{h}$ currently under consideration is updated as follows at step S35:

$$\mathbf{h}(p) = \mathbf{h}(p) - i \cdot d \tag{65a}$$

Equation (65a) means that the imaginary part of $\mathbf{h}(p)$ is decreased by d.

All elements of the auxiliary vector $\mathbf{Q}$ are also updated at step S36, in accordance with equation (66a).

$$\mathbf{Q}(r) = \mathbf{Q}(r) - d \cdot i \cdot \mathbf{R}(p,r), \forall r : 1 \le r \le N \tag{66a}$$

If arg = 1, 2, 3, or 4, Flag is set to '1' at step S13, control then passes to step S14 and the algorithm proceeds as described above with reference to Figure 1.

If arg = 5, no updates are made to $\mathbf{h}$ or $\mathbf{Q}$ and control passes to step S14, and the algorithm proceeds as described above.

Step S27 described above, requires the minimum of five values to be identified so as to determine the next course of action:

$$\min\left\{\mathrm{Re}\{\mathbf{Q}(p)\},-\mathrm{Re}\{\mathbf{Q}(p)\},\mathrm{Im}\{\mathbf{Q}(p)\},-\mathrm{Im}\{\mathbf{Q}(p)\},-\mathbf{R}(p,p)\cdot\frac{d}{2}\right\}$$

a method for finding this minimum, and efficiently identifying the correct action (at step S28 of Figure 9), is now described with reference to the flowchart of Figure 10

The algorithm takes as input two values *a* and *b*. *a* is input at a step S101 and is set to be equal to the real part of **Q**(p), and *b* is input at a step S102 and is set to be equal to the imaginary part of **Q**(p).

A decision block S103 determines whether or not *a* is greater than 0. If *a* is positive, *a* is set to be equal to the negative of its current value at a step S104, and a variable Ja is set to '1' at step S105. If *a* is not positive, the variable Ja is set to '0' at step S106.

b is processed in a similar manner. Step S107 checks whether *b* is positive. If *b* is positive, it is set to the negative of its current value at step S108, and a variable Jb is set to '1' at step S109. If *b* is not positive, Jb is set to '0' at step S110.

Thus after execution of steps S101 to S110, Ja is set to '1' if the input value of *a* was positive, and '0' if the input value of *a* was not positive. Jb is similarly set. Given the action of steps S104 and S108, it can be seen that both *a* and *b* will now not be positive.

At step S111 a check is made to determine whether *a* is greater than *b*. If this condition is true a variable J1 is set to be equal to Jb, a variable J2 is set to '0' and a variable c is set to be equal to b. This is accomplished by step S112. If the condition of step S111 is false, a variable J1 is set to be equal to Ja, a variable J2 is set to '0' and c is set to be equal to a. This is accomplished by step S113.

At step S114, a check is made to determined whether or not *c* is greater than -**R**(*p*,*p*)*d*/2. If this condition returns true, a variable J3 is set to be equal to '1' at step

S115. If this condition is false, the variable J3 is set to be equal to '0' at the step S116.

The method of Figure 10 is such that after execution of all steps, the variables J1, J2 and J3 are set as follows:

$J3 = 1 \Rightarrow$ no update necessary
$J3 = 0 \Rightarrow$ update necessary

$J2 = 1 \Rightarrow$ update to imaginary part necessary
$J2 = 0 \Rightarrow$ update to real part necessary

$J1 = 1 \Rightarrow$ update should comprise subtraction
$J1 = 0 \Rightarrow$ update should comprise addition

It will be appreciated that the modifications made to the algorithm of Figure 1, as illustrated in Figures 6 and 8 can similarly made to the algorithm of Figure 9. For example, the algorithm of Figure 9 can be implemented using an array *delta*, similar to that used in Figure 6, where:

$$\begin{aligned} &delta[1] = d; \\ &delta[2] = -d; \\ &delta[3] = d \cdot i; \\ &delta[4] = -d \cdot i; \end{aligned} \tag{67}$$

Having established *delta* in this way, the algorithm can proceed by simply adding the appropriate element of *delta* to the appropriate element or elements of **h** or **Q** as described above. Figure 11 shows this variant of the algorithm of Figure 9 where steps S28 to S36 are replaced by steps S37 to S39. It will be appreciated that additionally the array *delta* must be initialised and updated after each update of d in accordance with equation (67). This is not shown in Figure 11. Figure 12 shows a MATLAB program implementing the algorithm illustrated in Figure 11.

The description presented above has illustrated various implementations of algorithms in accordance with the invention. It will be appreciated that various amendments can be made to these algorithms without departing from the invention.

For example, in the description set out above execution ends when the number of iterations *it* for any particular bit *m* reaches a predetermined limit *Nit*. It will be appreciated that execution need not end in this circumstance. Instead, a timer *t* may be set to '0' each time *m* is updated, and execution can end if this timer exceeds a predetermined time threshold.

The vectors **h** and **Q** need not necessarily be initialised as indicated above. Indeed, the initial value for **h** should usually be substantially centrally positioned in the range $-H$ to H in which solutions are being sought, so as to obtain quick convergence. In some embodiments of the present invention the auxiliary vector need not be created. Instead the vector β is used directly, and is updated in a manner similar to the manner described above for vector **Q**, although it will be appreciated that different updates will be required. Suitable updates for such embodiments of the invention are set out in the derivation of the algorithm presented below.

It has been described above that *d* is updated by dividing the previous value of *d* by two. This is preferred because considerable benefits are achieved because computations involving multiplication of *d* can be carried out using efficient bit shift operations in place of relatively inefficient multiplication operations. However, it will be appreciated that alternative methods for updating *d* may be used in some embodiments of the invention. For example, if *d* is updated by division by a power of two such as four or eight, computations can still be efficiently implemented by carrying out two or three bit shifts instead of the single bit shifts required when *d* is updated by division by two.

In preferred embodiments of the present invention, each value of the solution vector **h** is represented by a fixed point binary word. This is particularly beneficial given that mathematical operations can typically be carried out more efficiently using

fixed point arithmetic. Furthermore, a fixed point representation is likely to be acceptable because the different unknown variables are likely to have an approximately equal magnitude.

In circumstances where a fixed point representation is inappropriate for the solution vector values, a conventional floating point representation can be used. Although the algorithm will operate more slowly with floating point values than with fixed point values, the algorithm still offers very favourable performance as compared with other methods for solving linear equations.

It will be appreciated that the algorithm described above can be implemented in software or hardware. A software implementation will typically comprise appropriate source code executing on an appropriate microprocessor. For example, as shown in Figures 7 and 12, code can be created in MATLAB which is compiled to create object code which is specified in the instruction set of a microprocessor. Although MATLAB provides a convenient implementation for the algorithms of the invention, it will be appreciated that the algorithms could instead be coded in any one of the large number of widely available computer programming languages.

A hardware implementation of the algorithm can either be provided by configuration of appropriate reconfigurable hardware, such as an application specific integrated circuit (ASIC), field programmable gate array (FPGA), or a configurable computer (CC), or alternatively by a bespoke microprocessor built to implement the algorithm. Figures 13 to 18 illustrate a device and microprocessor configured to solve linear equations.

Figure 13 schematically illustrates the general architecture of a device configured to solve linear equations. The device comprises a host processor 7 which is a general purpose microprocessor of known form configured to control operation of the device by running appropriate program code. This program code can be stored on a non volatile storage device 8 (e.g ROM or FLASH memory) and read by the general purpose microprocessor 7 when it is to be executed. Alternatively the program code

can be copied to a volatile memory 9 (e.g RAM) prior to execution. In order to solve linear equations using the method of the invention, the device comprises an equation solving microprocessor 10, operation of which is controlled by the host processor 7. The aforementioned components of the device are connected together by a host bus 11.

Figure 14 illustrates the architecture of the equation solving microprocessor 10 in further detail. The equation solving microprocessor 10 comprises a controller 12, the function of which is to control operation of the equation solving microprocessor 10. The equation solving microprocessor 10 further comprises a h-block 13, which stores and updates the solution vector **h** of the algorithm, a Q-block 14 which stores and updates the auxiliary vector **Q** of the algorithm, and a R-block 15 which stores the coeffient matrix **R** used by the algorithm. The equation solving microprocessor 10 also comprises a minimisation block 16 which finds the minimum of a number of values. Components of the equation solving microprocessor 10 are connected together by an internal bus 17, along which control instructions and data can pass.

Figure 15 shows the structure of R-block 15 in further detail. The R-block 15 comprises a storage element 18 which stores the elements of the coefficient matrix **R**. The R-block also comprises multiplexers 19, 20 and 21 which generate a column address signal 22, and a row address signal 23 for reading data from and writing data to the storage element 18 in the manner described below. Finally, the R-block comprises a bit-shift element 24 which can left-shift a value presented at its input by a value determined by the current value of d.

The multiplexers 19, 20, 21 generate address signals as follows. The row multiplexer 20 and the column multiplexer 21 each have selection inputs connected to a common input line 25 which can carry an initialisation signal *Init* which is typically received from the controller 12 (Figure 14). If an initialisation signal is received, this indicates that data is to be written to the storage element 18 representing the coefficients of the equations which are to be solved. In this case the row multiplexer 20 should generate a row address 23 which is equal to the input row address 26, and

similarly the column multiplexer 21 should generate a column address 22 which is equal to the input column address 27. During initialisation, the row address 26 and column address 27 will typically count through elements of the matrix **R** writing data provided to the storage element 18 on input line 28 to the appropriate element of the storage element 18.

When initialisation has been completed, the *Init* signal will no longer be received by the selection input of the multiplexers 20, 21, and therefore the row address is determined by the input 29 to the row multiplexer 20 and the column address is determined by the input 30 to the column multiplexer 21. The input 30 is the value p of the algorithm as described above. The input 29 is connected to the update multiplexer 19 whose output is dependent upon whether values of **R** are being read for purposes of analysis (e.g step S7 of Figure 1) or update of **Q** (e.g. steps S10 and S12 of Figure 1). The update multiplexer 19 has a selection input 31 which indicates either update or analysis. This can conveniently be achieved, for example, by a single bit input where '0' indicates update and '1' indicates analysis.

If **R** is being read for purposes of analysis (i.e. the selection input 31 is set to '1'), then the element $\mathbf{R}(p,p)$ is required (see step S7 of Figure 1). Therefore, update multiplexer output 29 is set to be equal to the input 32 which is equal to p, and the row multiplexer 20 subsequently generates a row address 23 equal to p.

If **R** is being read for purposes of update of the auxiliary vector **Q** (i.e. selection input is set to '0') $\mathbf{R}(p,r)$ is required (see steps S10 and S12 of Figure 1). Therefore the update multiplexer output 29 is equal to the input 33 which is set to be equal to r, and the row multiplexer 20 subsequently generates a row address 23 equal to r.

In either case, the column multiplexer 21 generates a column address 22 which is equal to p.

From the preceding description, it can be seen that the multiplexers 19, 20 and 21, act together to ensure that the correct row address 23 and column address 22 are sent

to the storage element 18. Having read the appropriate element of **R** from the storage element 18, the output needs to be multiplied by d regardless of whether it is being used for update or analysis (see steps S7, S10 and S12 of Figure 1). As described above, this can be achieved by a bit shift (assuming that d is a power of 2), the number of bits to shift being determined by the expression:

$$\log_2 d \tag{68}$$

which is passed to an input 34 of the bit shift block 24. The output 35 from the bit shift block 24 is then correctly set as follows:

$$d \cdot \mathbf{R}(p,r), \text{ if update} \tag{68a}$$

$$d \cdot \mathbf{R}(p,p), \text{ if analysis} \tag{68b}$$

Figure 16 illustrates the h-block 13 of Figure 14 in further detail. The h-block comprises a storage element 36 for storing elements of the solution vector **h**, an update/reading multiplexer 37, and an adder-subtractor 38.

To initialise elements of the solution vector, an initialisation signal 39 is input to the storage element 36. Upon receipt of this signal, all elements of the solution vector **h** are initialised to '0'.

The update/reading multiplexer 37 sends an appropriate address 39a to the storage element 36. The address 39a sent to the storage element 36 is determined by the signal provided to a selection input 40 of the multiplexer 37. If the selection input 40 is set to update, the address p provided at input 41 becomes the address 39a.

If the selection input 40 is set to read (i.e. the values of **h** are being read to obtain the solution of the equations), a read address 42 input to the multiplxer 37 becomes the address 39. In the case of a read operation, the input read address will count through

all elements of **h** in turn, and each element will be transmitted to the internal bus 17 of the equation solving microprocessor 10 in turn.

In the case of an update operation, a single value of p is provided to the multiplexer 37, and an input signal I_1 is provided to the adder/subtractor 38. The signal I_1 indicates whether the element of the solution vector **h**(p) is to be updated by adding d or subtracting d. Upon receipt of the address 39, the storage element 36 outputs the appropriate element to the adder/subtractor 38 along a line 44. By analysing the signal I_1 the adder/subtractor 38 can determine whether its output should be set to **h**(p)+d or **h**(p) – d. The appropriate expression is calculated, and written back to the storage element 36 along a line 45.

Figure 17 shows how the h-block 16 can be implemented when the algorithm is used to solve equations having complex values (for example using the algorithm of Figures 10, 11 or 12). It can be seen that in addition to the components illustrated in Figure 16, the update/read multiplexer 37 has an additional input 46 which carries a signal I_2. This signal is sent to the storage element 36 along a line 47 along with the address p when the selection input is set to update. The signal I_2 indicates whether the update should be made to the real part or the imaginary part of **h**(p).

Figure 18 illustrates the Q-block 14 of Figure 14 in further detail. It can be seen that the Q-block comprises a storage element 48 for storing the vector Q, an address multiplexer 49, a data multiplexer 50 and an adder/subtractor 51.

The address multiplexer 49 has a selection input 52 which selects update, analysis, or initialisation mode. The address multiplexer 49 also has three data inputs, a first data input 53 carries the value r used in the algorithm, a second data input 54 carries the value p used in the algorithm, and a third data input 55 carries initialisation address data.

When the selection input 52 of the address multiplexer 49 is set to initialise, an output 56 of the address multiplexer carries the initialisation address supplied at the

third input 55 of the address multiplexer. When the selection input 52 is set to update, the output 56 carries the value r supplied at the first input 53 of the address multiplexer 49. When the selection input 52 is set to analysis, the output 56 of the address multiplexer 49 is set to the value p supplied at the second input 54 to the address multiplexer 49. The output 56 of the address multiplexer is passed to the storage element 48 as an address for reading or writing data.

Data is written to the storage element 48 only in the update and initialisation modes. Therefore, the data multiplexer 50, has a selection input 57 having two recognised values, update and initialise (it will be appreciated that in practice this selection input may be common with the selection input 52 of the address multiplexer 49, although the behaviour of the data multiplexer is not well defined when the selection input is set to analysis).

The data multiplexer 50 comprises a first data input 58 carrying initialisation data (elements of the vector β) and a second data input 59 carrying update data. When the selection input 57 of the data multiplexer 50 is set to initialise, data from the first data input 58 is provided at the output 60 of the data multiplexer 50. When the selection input 57 is set to update, data from the second data input 59 is provided at the output 60.

In operation, when the selection inputs of both multiplexers 49, 50 are set to initialise, a sequence of addresses (counting through all elements of the vector $\mathbf{Q}$) are provided at the third data input 55 of the address multiplexer 49. In synchronisation with these addresses, the appropriate data is provided at the first data input 58 of the data multiplexer 50. The data and addresses are provided to the storage element 48, and the storage element is then set to contain the appropriate elements of the vector $\mathbf{Q}$.

When the selection inputs of both multiplexers 49, 50 are set to update, the address r is provided to the storage element 48 by the multiplexer 49. The appropriate entry from the storage element 48, $\mathbf{Q}(r)$ is provided at an output 61 of the storage element,

and passed to a first input 62 of the adder/subtractor 51. A second input 63 of the adder/subtractor is set to $d \cdot \mathbf{R}(p,r)$ (provided by the R-block 15 of Figure 14). The adder/subtractor also includes an input 64 carrying a signal I_1 which indicates whether the adder/subractor 51 should perform an addition or subtraction operation. On receipt of appropriate data at its inputs an output 65 of the adder/subtractor is set to the output of the operation preformed, and this data is fed to the data multiplexer 50. Given that the selection input 57 is set to update, the data provided at the input 59 is provided to the output 60 of the data multiplexer, and from there to the storage element 48 where it is written to the element of **Q** specified by the address 56.

Where the algorithm is being implemented to solve complex valued equations, the adder/subtractor comprises a further input 66 carring an signal I_2 which indicates whether the update should be applied to the real part or the imaginary part of the appropriate element of the vector **Q**.

In analysis mode, the Q-block is required to provide a current value of $\mathbf{Q}(p)$, with no update functions being needed. The data multiplexer 50 is therefore not involved with the analysis operation. The address multiplexer 49 provides the address p provided at its second input 54 to the storage element 48. The appropriate value of $\mathbf{Q}(p)$ is read from the storage element 48 and provided at the output 61 of the storage element 48.

Figure 19 provides an implementation for the minimisation block 16 of Figure 14. This implementation corresponds to that illustrated in Figure 10, and is therefore configured for use in algorithms solving equations having complex valued solutions. The minimisation block comprises first and second converter blocks 67, 68, first and second comparators 69, 70 and first and second multiplexers 71, 72.

Input data is provided to the first and second converter blocks 67, 68. The first converter block 67 takes as input the real part of a value $\mathbf{Q}(p)$, and the second converter block 68 takes as input the imaginary part of the value $\mathbf{Q}(p)$. The converter blocks 67 and 68 provide two outputs, a first output 67a, 68a indicates the sign of the

input data, and second output 67b, 68b indicates the modulus of the input data. The modulus outputs 67b, 68b are input to the first comparator 69, which generates a signal I_2 which at an output 73. The signal I_2 is defined as being '0' if the value provided at output 67b is greater than that provided at 68b, and '1' if the value provided at output 68b is greater than or equal to that provided at output 67b.

The values provided at outpus 67b and 68b are also provided to the first multiplexer 71, along with the output value 73 created by the first comparator 69. The output 73 of the first comparator acts as a selection input to the first multiplexer 71. The first multiplexer 71 then acts to provide the larger of the values 67b, 68b at its output 74.

The output 74 of the multiplexer 71 is input to the second comparator 70, along with the value $d \cdot \mathbf{R}(p,p)$ at an input 75, which is computed by the R-block 16 of Figure 15. The second comparator 70 then produces an output 76 which is a signal I_3 which is set such that I_3 is set to '0' if the input 74 greater than the input 75, and '1' if the imputer 75 is greater than or equal to the input 74.

The second multiplexer 72 is a bit multiplexer taking as input the signs of the input data generated by the converter blocks 67, 68. In dependence upon the value of the signal I_2 output from the first comparator 69, the multiplexer generates an output 77 which is a signal I_1.

The signals I_1, I_2 and I_3 can together be used to determine what update (if any) should be made to the elements of **Q** and **h**. I_3 indicates whether or not the current iteration is successful. If I_3 is equal to '0', the element $\mathbf{h}(p)$ is updated, and all elements of **Q** are updated. If I_3 is equal to '1', no updates are necessary.

I_2 indicates whether the real or the imaginary part of the appropriate elements should be updated. If I_2 is equal to '0' the real part of the appropriate values is updated, while if I_2 is equal to '1' the imaginary part of the appropriate values is updated. I_1 indicates whether the update should comprise addition or subtraction. If I_1 is set to '0', addition is used, and if I_1 is set to '1', subtraction is used.

Having described the structure and function of the individual components of the equation solving microprocessor 10 (Figure 14), operation of the processor is now described.

To perform initialisation, the equation solving microprocessor receives a signal to cause initialisation, for example from the host processor (Figure 13). The controller 12 of the equation solving microprocessor then generates an appropriate internal initialisation signal which is communicated to all blocks of the equation solving microprocessor, via the internal bus 17. Upon receipt of this signal the h-block 13, the Q-block 14 and the R-block 15 perform initialisation as described above. The data required for initialisation may be located in a predefined read only memory, or an address where the data is to be found may be provided to the controller 12 for onward transmission to the appropriate block. Parameters used by the algorithm (e.g. *N, M* and N*it*) can either be provided to the controller, or alternatively specified within the microprocessor 10.

When initialisation is complete, the controller begins executing an algorithm in accordance with the invention using the blocks of the microprocessor 10 to update values of **h** and **Q** as appropriate. In some embodiments of the invention data that is required to be passed between blocks of the microprocessor is passed directly between blocks, under the control of the sending and/or receiving block. In alternative embodiments of the invention all data is passed between blocks as directed by the controller 12. When the controller determines that the equations have been solved, such that the vector **h** contains the solution of the equations, the vector **h** can then be copied from the storage element 36 (Figures 16 and 17) to an appropriate location within the device, where the solutions can be used as required.

It will be appreciated that although a specific hardware implementation of algorithms of the invention has been described above, numerous modifications could be made to the implementation while remaining within the scope and spirit of the invention. Such modifications will be readily apparent to those of ordinary skill in the art.

The preceding description has described algorithms in accordance with the invention, and hardware suitable for implementing such algorithms. The mathematical basis of algorithms in accordance with the invention is now described.

To aid understanding of the invention, the known co-ordinate descent optimisation method for minimising a function of many variables is presented. The co-ordinate descent optimisation method seeks to find:

$$\min\{J(\mathbf{h})\} \tag{69}$$

where J is a function of many variables; and

$\mathbf{h}$ is an N-dimensional vector containing the variables.

Thus we want to find the value of $\mathbf{h}$ when the function J is a minimum.

In many circumstances the elements of $\mathbf{h}$ have a maximum amplitude H. Therefore, the minimisation problem is considered for a subset of values of $\mathbf{h}$ defined by equations (70) and (71).

$$\mathbf{h} \in \mathbf{U} \tag{70}$$

where:

$$\mathbf{U} = \left\{\mathbf{h}(m) : 1 \le m \le N \wedge \left|\mathbf{h}(m)\right| < H\right\}; \tag{71}$$

where:

$\mathbf{h}(m)$ are elements of the vector $\mathbf{h}$ and H is a known number such that $H > 0$.

Define a vector $\mathbf{e}_i$ where the ith coordinate is '1', and all other coordinates are '0'. Let $\mathbf{h}_0$ be an initial value of the solution vector $\mathbf{h}$ and let α_0 be an initial value of the step size parameter (that is d in the algorithms described above).

$\mathbf{h}_k$ is defined to be the value of the solution vector $\mathbf{h}$ after some number of iterations k, where k is a positive integer. α_k is the step size parameter after k iterations.

A vector $\mathbf{p}_k$ is also defined, according to the equation:

$$\mathbf{p}_k = e_{i_k} \tag{72}$$

where:

$$i_k = k - N(\mathrm{div}(K, N)) + 1 \tag{73}$$

where div (A,B) is a function whose result is defined as the integer part of result of dividing A by B.

It can be seen from equation (73) that i_k will repeatedly count from 1 to N as k is increased. This results in values of $\mathbf{p}$ cycling through $\mathbf{e}_1$ to $\mathbf{e}_N$:

$$\mathbf{p}_1 = \mathbf{e}_1, \mathbf{p}_2 = \mathbf{e}_2, \ldots, \mathbf{p}_N = \mathbf{e}_N, \mathbf{p}_{N+1} = \mathbf{e}_1, \mathbf{p}_{N+2} = \mathbf{e}_2, \ldots, \mathbf{p}_{2N} = \mathbf{e}_N \tag{74}$$

The value of the function J is calculated at the point $\mathbf{h} = \mathbf{h}_k + \alpha_k \mathbf{p}_k$ and two conditions are checked:

$$\mathbf{h}_k + \alpha_k \mathbf{p}_k \in \mathbf{U} \tag{75}$$

$$J(\mathbf{h}_k + \alpha_k \mathbf{p}_k) < J(\mathbf{h}_k) \tag{76}$$

If the conditions of equations (75) and (76) are satisfied, then the solution vector **h** is updated as follows:

$$\mathbf{h}_{k+1} = \mathbf{h}_k + \alpha_k \mathbf{p}_k \tag{77}$$

Given that that update involves a scalar multiple of the vector $\mathbf{p}_k$ and given also that $\mathbf{p}_k$ is a vector containing only a single non-zero element, it can be seen that equation (77) means that $\mathbf{h}_{k+1}$ will be identical to $\mathbf{h}_k$ save for a single element.

The step size parameter is not updated at this stage, as indicated by equation (78)

$$\alpha_{k+1} = \alpha_k \tag{78}$$

If the conditions of equations of (75) and (76) are not satisfied, the value of the function J is calculated at the point $\mathbf{h} = \mathbf{h}_k - \alpha_k \mathbf{p}_k$ and two conditions are again checked:

$$\mathbf{h}_k - \alpha_k \mathbf{p}_k \in \mathbf{U} \tag{79}$$

$$J(\mathbf{h}_k - \alpha_k \mathbf{p}_k) < J(\mathbf{h}_k) \tag{80}$$

If the conditions of equations (79) and (80) are satisfied, then the solution vector **h** is updated as follows:

$$\mathbf{h}_{k+1} = \mathbf{h}_k - \alpha_k \mathbf{p}_k \tag{81}$$

Again, given that that update involves a scalar multiple of the vector $\mathbf{p}_k$ and given also that $\mathbf{p}_k$ is a vector containing only a single non-zero element, it can be seen that equation (81) means that $\mathbf{h}_{k+1}$ will be identical to $\mathbf{h}_k$ save for a single element.

The step size parameter is not updated at this stage, as indicated by equation (82)

$$\alpha_{k+1} = \alpha_k \tag{82}$$

The *k*th iteration is considered to be successful if either the conditions of equations (75) and (76) or equations (79) and (80) are satisfied. If neither of these conditions are satisfied, the iteration is considered to be unsuccessful.

α_k is updated according to equation (83):

$$\alpha_{k+1,} = \begin{cases} \lambda\alpha_k, \text{if } i_k = N \wedge \mathbf{h}_k = \mathbf{h}_{k-N+1} \\ \alpha_k, \text{otherwise} \end{cases} \tag{83}$$

where λ is a parameter of the algorithm, and is such that $0 < \lambda < 1$

The condition of equation (83) means that if the last N iterations involve no successful updates (i.e. the value of $\mathbf{h}$ has not changed), the step size parameter is updated by multiplication by λ. If however there is at least one update during the previous N iterations, the step size parameter is not updated. It should be recalled that N is defined to be the number of elements in the solution vector $\mathbf{h}$, and it can therefore be seen that $\mathbf{h}$ is updated only when all elements have been processed and no update has occurred.

The method described above is generally known as co-ordinate descent optimisation. It is known that for some arbitrary function J, the method converges to find the value of $\mathbf{h}$ for which the function is a minimum, providing that the function J is convex and differentiable on $\mathbf{U}$, providing that $\alpha_0 > 0$, and $0 < \lambda < 1$ and providing that $\mathbf{h}_0 \in \mathbf{U}$. This result is shown, for example, in Vasiliev, F.P.: "Numerical methods for solutions of optimisation problems", Nauka, Moscow 1988 (published in Russian), the contents of which is herein incorporated by reference.

It is often necessary to solve the linear least squares problem. The linear least squares problem is concerned with the minimisation of the function J specified in equation (84):

$$J(\mathbf{h}) = |\mathbf{Zh} - \mathbf{d}|^2 = (\mathbf{Z}h - \mathrm{d})^T(\mathbf{Zh} - \mathbf{d}) \tag{84}$$

with respect to an unknown vector $\mathbf{h}$, where $\mathbf{Z}$ is a known M x N matrix, $\mathbf{d}$ is a known vector of length N, and "T" denotes the transpose of a matrix.

This is discussed in Sayed, A.H., and Kailath, K.: "Recursive least-squares adaptive filters", The Digital Signal Processing Handbook, CRC Press, IEEE Press 1998, pages 21.1 to 21.37, which discussion is herein incorporated by reference.

It can be shown that the minimisation of the function of equation (84) is equivalent to minimisation of a quadratic function.

Equation (84) can be rewritten as:

$$J(\mathbf{h}) = \mathbf{h}^T\mathbf{Z}^T\mathbf{Z}h - \mathbf{h}^T\mathbf{Z}^T\mathbf{d} - \mathbf{d}^T\mathbf{Z}h + \mathbf{d}^T\mathbf{d} \tag{85}$$

by multiplying out the bracketed expressions of equation (84).

Given that the purpose of the method is to minimise J with respect to $\mathbf{h}$ it can be concluded that the term $\mathbf{d}^T\mathbf{d}$ of equation (85) will not effect the minimisation process, and therefore can be removed from the expression without affecting the minimum value. Therefore minimisation of equation (85) is equivalent to minimisation of equation (86):

$$J(\mathbf{h}) = \mathbf{h}^T\mathbf{Z}^T\mathbf{Z}h - \mathbf{h}^T\mathbf{Z}^T\mathbf{d} - \mathbf{d}^T\mathbf{Z}h \tag{86}$$

A matrix $\mathbf{R}$ is defined according to equation (87):

$$\mathbf{R} = \mathbf{Z}^T\mathbf{Z} \tag{87}$$

A matrix β is defined according to equation (88):

$$\beta = \mathbf{Z}^T\mathbf{d} \tag{88}$$

Equation (86) can then be rewritten using $\mathbf{R}$ and β as shown in equation (89):

$$J(\mathbf{h}) = \mathbf{h}^T\mathbf{R}\mathbf{h} - \mathbf{h}^T\beta - \beta^T\mathbf{h} \tag{89}$$

Simplifying this expression yields:

$$J(\mathbf{h}) = \mathbf{h}^T\mathbf{R}\mathbf{h} - 2\mathbf{h}^T\beta \tag{90}$$

The expression $\mathbf{h}^T\mathbf{R}\mathbf{h}$ can be rewritten in terms of the elements of $\mathbf{h}$, and $\mathbf{R}$ as follows, using the definitions of matrix multiplication, and the matrix transpose operation:

$$\mathbf{h}^T\mathbf{R}\mathbf{h} = \sum_{m=1}^{N}\sum_{n=1}^{N}\mathbf{R}(m,n)\mathbf{h}(m)\mathbf{h}(n) \tag{91}$$

Similarly the expression $\mathbf{h}^T\beta$ can be written in form of equation (92):

$$\mathbf{h}^T\beta = \sum_{n=1}^{N}\beta(n)\mathbf{h}(n) \tag{92}$$

Substituting equations (91) and (92) into equation (90) yields:

$$J(h) = \sum_{m=1}^{N}\sum_{n=1}^{N}\mathbf{R}(m,n)\mathbf{h}(m)\mathbf{h}(n) - 2\sum_{n=1}^{N}\beta(n)\mathbf{h}(n) \tag{93}$$

It can be seen that equation (93) is a quadratic function of $\mathbf{h}$. Thus it can be seen that solving the linear least squares problem is equivalent to minimisation of the function

of equation (93). Furthermore, it is also known that solving a system of linear equations of the form of equation (1):

$$\mathbf{Rh} = \beta \tag{1}$$

is equivalent to minimisation of the function of equation (93), for any set of normal linear equations. This is explained in, for example, Moon, Todd K., and Stirling, Wynn C.: "Mathematical methods and algorithms for signal processing", Prentice Hall, 2000, section 3.4. "Matrix representations of least-squares problems", pages 138-139. This explanation is incorporated herein by reference.

Given that many sets of linear equations occurring the electronics and physics are normal linear equations, minimisation of the function of equation (93) has wide applicability in solving linear equations.

The explanation presented above has set out a method for minimising a function J using the co-ordinate descent optimisation method. The material presented above has also set out the relationship between the minimisation of equation (93) and a set of linear equations of the form of equation (1).

The present inventors have surprisingly discovered that applying the known co-ordinate descent method to the minimisation of equation (93) provides a particularly efficient method for solving linear equations.

Minimisation of the function of equation (94) is considered:

$$J(h) = \frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N}\mathbf{R}(m,n)\mathbf{h}(m)\mathbf{h}(n) - \sum_{n=1}^{N}\beta(n)\mathbf{h}(n) \tag{94}$$

This minimisation process finds values for the elements of **h** which minimise the function $J(\mathbf{h})$. The matrix **R** and the vector β are known. It is known that the function of equation (94) is convex and differentiable. This is shown in Vasiliev, F.P.: "Numerical methods for solutions of optimisation problems", Nauka, Moscow 1988 (published in Russian), page 345, which explanation is incorporated herein by reference. Therefore, as explained above, the co-ordinate descent optimisation method can be used to find the minimum value of the function J.

During operation of the co-ordinate descent optimisation method, the following expressions are computed:

$$\Delta J(\mathbf{h}_k) = J(\mathbf{h}_k + \alpha_k \mathbf{e}_{i_k}) - J(\mathbf{h}_k) \tag{95}$$

and

$$\Delta J(\mathbf{h}_k) = J(\mathbf{h}_k - \alpha_k \mathbf{e}_{i_k}) - J(\mathbf{h}_k) \tag{96}$$

It can be seen that equation (95) relates to the condition of equation (76) set out above, while equation (96) relates to the condition of equation (80) set out above.

The inequality of equation (97) is also considered:

$$\Delta J(\mathbf{h}_k) < 0 \tag{97}$$

It can be recalled that the values of the vector $\mathbf{e}_{i_k}$ has elements defined by:

$$[e_{i_k}]_i = \begin{cases} 1 \text{ if } i = i_k \\ 0 \text{otherwise} \end{cases} \tag{98}$$

Also, it is known that the matrix **R** is symmetric, given that the system of equations is normal. Therefore, substituting equation (94) into equation (95) yields:

$$\Delta J(\mathbf{h}_k) = \frac{\alpha_k}{2}\left[-2\beta(i) + 2\sum_{m=1}^{N} h^{(k)}(m)\mathbf{R}(m,i) + \alpha_k \mathbf{R}(i,i)\right] \tag{99}$$

where $h^{(k)}(m)$ are elements of the vector $\mathbf{h}_k$.

If a vector $\mathbf{Q}$ is defined as:

$$\mathbf{Q}(i) = -2\beta(i) + 2\sum_{m=1}^{N} \mathbf{h}^{(k)}(m)\mathbf{R}(m,i) \tag{100}$$

then equation (99) can be written as:

$$\Delta J(\mathbf{h}_k) = \frac{\alpha_k}{2}\left[\mathbf{Q}(i) + \alpha_k \mathbf{R}(i,i)\right] \tag{101}$$

Given that $\alpha_k > 0$, from equation (101), equation (95) can be rewritten as:

$$\alpha_k \mathbf{R}(i,i) < -\mathbf{Q}(i) \tag{102}$$

Similarly, equation (96) can be rewritten as:

$$\alpha_k \mathbf{R}(i,i) > \mathbf{Q}(i) \tag{103}$$

An auxiliary vector $\mathbf{Q}_k$ is defined as the vector $\mathbf{Q}$ after the kth iteration. If the $(k+1)$th iteration is not successful, then elements of $\mathbf{h}$ and $\mathbf{Q}$ can be updated as follows:

$$\mathbf{h}_{k+1} = \mathbf{h}_k \tag{104}$$

and

$$\mathbf{Q}_{k+1} = \mathbf{Q}_k \tag{105}$$

If the $(k+1)$th iteration is successful, then elements of **h** and **Q** are updated as follows:

$$\mathbf{h}^{(k+1)}(i) = \mathbf{h}^{(k)}(i) \pm \alpha_k \tag{106}$$

$$\mathbf{Q}^{(k+1)}(n) = \mathbf{Q}^{(k)}(n) \pm 2\alpha_k R(n,i), n = 1,...,N \tag{107}$$

The vector **h** can be initialised to a vector $\mathbf{h}_0$ where

$$\mathbf{h}_0(n) = 0,\ n = 1,...,N \tag{108}$$

Then, from equation (100), elements of **Q** are initialised as follows:

$$\mathbf{Q}_0(n) = -2\beta(n),\ n = 1,...,N \tag{109}$$

That is , each element of **Q** is set to be the negative of the corresponding element of β multiplied by two.

Thus, the solution vector **h** is initialised to contain all '0' values, while the auxiliary vector **Q** is initialised to be the negative of the vector β.

As described above, multiplication operations may be avoided by setting H in accordance with equation (110):

$$H = 2^{P+M_b} \tag{110}$$

where M_b is a positive integer, and P is any integer.

α_0 is initialised to be $\frac{H}{2}$ and λ is initialised to be $\frac{1}{2}$.

The multiplications described above can then be replaced by bit shift operations.

The algorithms described thus far have used an auxiliary vector **Q** which is intialised in accordance with equation (4):

$$\mathbf{Q} = -\beta \tag{4}$$

However, some embodiments of the invention use β itself as an auxiliary vector. In such embodiments of the invention, the auxiliary vector update rule of equation (107) above becomes:

$$\beta^{(k+1)}(n) = \beta^{(k)}(n) \mp \alpha_k \mathbf{R}(n,i), n = 1,\ldots,N \tag{111}$$

Similarly, the inequalities of equations (102) and (103) become:

$$\alpha_k \mathbf{R}(i,i) < 2\beta(i) \tag{112}$$

and:

$$\alpha_k \mathbf{R}(i,i) > -2\beta(i) \tag{113}$$

Figure 19a shows MATLAB source code for an algorithm which uses β in place of **Q**. This algorithm is based upon that of Figure 7, but the code has been amended as set out above.

The step size parameter α_k is updated if N consecutive iterations are not successful. At every update of α_k, α_k is decreased by a factor of two. The algorithm described above is therefore referred to as the dichotomous coordinate descent algorithm for the solution of linear equations.

From the description set out above, it can be observed that the algorithms of the invention solve linear equations by minimisation of an appropriate quadratic

function. It has been explained above that it is known that such minimisation can be employed to solve normal linear equations. However, it should be noted that the present invention is not limited simply to normal linear equations, but is instead applicable to a wider class of linear equations.

In the methods described above, the elements of the solution vector **h** are analysed in a predetermined order (i.e. from element '1' to element *N*). However, it will be appreciated that elements of the solution vector **h** can be analysed in any convenient manner. For example, the values of **h** can be sorted on the basis of some corresponding auxiliary value (e.g. a corresponding element of the vector **Q**), and elements of the solution vector **h** can then be processed in that order. For some applications, ordering elements of the vector **h** in this way will provide a more rapid convergence, although this must of course be balanced against the computational cost of sorting the elements of **h**.

It has been explained above that the present invention can be usefully applied in any application in which it is necessary to solve linear equations. Two such applications are now described.

In a multiuser Code Division Multiple Access (CDMA) communications system, a plurality of users transmit data using a common collection of frequencies. A narrow band data signal which a user is to transmit is multiplied by a relatively broad band spreading code. Data is then transmitted using this broad band of frequencies. Each user is allocated a unique spreading code.

A receiver needs to be able to receive data transmitted by a plurality of users simultaneously, each user using his/her respective spreading code. The receiver therefore needs to implement functions which allow the spreading code to be removed from the received data to yield the originally transmitted data. Typically filters are used to extract the spreading code to obtain the transmitted data. It should be noted that the process is complicated by interfering signals from multiple users,

and also from different propagation paths which may be followed by different signals.

Figure 20 illustrates a receiver 80 suitable for use in a CDMA communications system. The receiver comprises a receiver circuit 81 including an antenna (not shown), an analog to digital convertor 82, a bank of filters 83a, 83b and 83c, an equation solving circuit 84 and a decision circuit 85.

Spread spectrum signals are received by the receiver circuit 81, and converted into digital data by the analog to digital converter 82. Digital data is then passed to all of the filters 83a, 83b, 83c. Each filter of the bank of filters 83a, 83b, 83c relates to a unique user, and has filter coefficients selected to match the spreading code of that user. It will therefore be appreciated that in practical embodiments a receiver will include more than three filters as is illustrated in Figure 20.

If a single signal is transmitted at any one time, and this signal travels between a sender and the receiver 80 by a direct path, the output of the filters alone should provide the data which the sender intended to transmit. However, in the more likely and more complicated situation where interference between signals occurs, the outputs of the filters alone will be inconclusive. However, it is known that solving an equation :

$$\mathbf{Rh} = \beta \tag{114}$$

where $\mathbf{R}$ is the cross correlation matrix of the spreading sequences of all users;
β is a vector containing the filter output values; and
$\mathbf{h}$ is a solution vector

will allow the originally transmitted data to be obtained..

In general, for a system involving N users, there will be N filters, and the vector β will therefore have length N, and the matrix **R** will have size N x N.

The cross correlation matrix **R** can be defined as

$$\mathbf{R} = \mathbf{S}^{\mathrm{T}}\mathbf{S} \tag{115}$$

where the matrix **S** contains the spreading codes, and is defined as follows:

$$\mathbf{S} = \begin{bmatrix} s_1(1) & s_2(1) & \cdots & s_k(1) \\ s_1(2) & s_2(2) & \cdots & s_k(2) \\ \vdots & \vdots & \ddots & \vdots \\ s_1(m) & s_2(m) & \cdots & s_k(m) \end{bmatrix} \tag{116}$$

where $s_j(i)$ denotes the ith element of the spreading code for user j.

As has been described above linear equations of the form shown in equation (114) can be solved using an algorithm in accordance with the invention. Therefore, the invention provides a novel multi user receiver apparatus, in which the equation (114) is solved as described above, thereby achieving the considerable performance benefits provided by solving equations in accordance with the invention.

The equation solver 84 of Figure 20 can either be implemented by means of a computer program carrying out the method of the invention executing on an appropriate microprocessor, or alternatively by means of hardware, such as that described above with reference with Figures 14 to 19.

The equation solver provides a vector **h** as output, and this is input to the decision circuit 85, which then determines the nature of the transmitted data.

It will be appreciated that the cross correlation matrix described with reference to equations (115) and (116) is merely exemplarily. Cross correlation matricies can be created in a variety of different ways which will be known to one skilled in the art. Regardless of how the cross correlation matrix is formed, a system of equations (114) is created which can be solved using methods in accordance with the present invention.

It will also be appreciated that in addition to the components illustrated in Figure 20 and described above, a CDMA receiver may require other components to function properly. The selection and use of these components will be readily apparent to those of ordinary skill in the art.

The algorithms of the invention can also be employed in adaptive filtering applications such as echo cancellation in a hands free communications system.

A system of interest in illustrated in Figure 21. A signal travels along input line 86 and is output through a loudspeaker 87. A further signal such a human voice (not shown) is passed to an input of a microphone 88. It is desirable that the signal at the output 89 of the microphone 88 contains only the human voice signal, and none of the signal output by the loudspeaker 87. However, in practice, some of the signal output by the loudspeaker 87 will be received by the microphone 88 such that the microphone output 89 comprises a combination of the human voice signal and part of the loudspeaker output signal (referred to as "the echo signal"). It is desirable to remove the echo signal present in the microphone output 89.

As shown in Figure 21, the echo cancellation apparatus comprises a filter 90, which is configured to provide an estimate 91 of the echo signal. This estimate 91 is subtracted from the microphone output signal 89 by a subtractor 92 . Therefore, if the echo is accurately estimated, an output 93 of the subtractor will be equal to the human voice signal.

The echo cancellation apparatus comprises a filter coefficient setting circuit 94, which takes as inputs a signal 86 which is input to the loudspeaker and the signal 89 which is output from the microphone. The circuit 94 should generate coefficients to allow the filter 90 to accurately model the echo signal.

Figure 22 shows the filter coefficient setting circuit 94 in further detail. It can be seen that the circuit comprises an auto correlation element 95, a cross correlation element 965 and an equation solver 97. The auto correlation element 95 finds the auto correlation of the signal 86. The cross correlation element 96 finds the cross correlation between the signal 86 and the signal 89. It is known that when a auto correlation matrix **R** and a cross correlation vector β have been generated, the optimal filter coefficients **h** can be found by solving the system of equations:

$$\mathbf{Rh} = \beta \tag{117}$$

where the auto correlation matrix **R** is generated according to the equation:

$$\mathbf{R}(m,n) = \sum_{t=1}^{T-|n-m|} x(t)x(t+|n-m|), \forall m,n : 1 < m < N \wedge 1 < n < N \tag{118}$$

where t=1,...,T are discrete time moments;

and the cross correlation vector β is generated according to the equation:

$$\beta(n) = \sum_{t=1}^{T-n-1} x(t)y(t+n-1)$$

where x is the loudspeaker input signal 86 and y is the microphone output signal 89.

An echo cancellation system operating in the manner described above is described in US5062102 (Taguchi).

Having generated a system of equations of the form of equation (117), algorithms in accordance with the present invention can be used to solve linear equations to determine a solution vector **h** containing optimal filter coefficients. Therefore, referring back to Figure 22, the equation solver 97 can either be a microprocessor executing code in accordance with one of the algorithms described above, or alternatively hardware suitably configured to implement a suitable algorithm.

It will be appreciated that although this application of the algorithm has been described with reference to echo cancellation, it is widely applicable in all cases where an adaptive filter is required, and where solving a system of linear equations yields appropriate filter coefficients. A suitable example system in which the invention could be beneficially employed is described in WO 00/38319 (Heping).

Applications of the invention to CDMA receivers, and echo cancellers have been described above. However, it will be appreciated that many other applications exist which can benefit by the improved efficiency with which linear equations can be solved in accordance with the invention. For example, the invention can be used in tomographic imaging systems, where a large system of linear equations is solved to generate an image.

Although the present invention has been described above with reference to various preferred embodiments, it will be apparent to a skilled person that modifications lie within the scope and spirit of the present invention.